CS 6375: Machine Learning - Project 1

Naive Bayes and Logistic Regression for Spam Classification

Navaneeta Padmakumar/NXP230016

10/03/2025

RESULTS TABLE

Table 1: Performance Metrics for All Algorithms

Dataset	Algorithm	Representation	Accuracy	Precision	Recall	F1-Score
enron1	Multinomial NB	BoW	0.9386	0.8903	0.9262	0.9079
enron1	Bernoulli NB	Bernoulli	0.8399	0.8654	0.6040	0.7115
enron1	Logistic Regression	BoW	0.9518	0.8944	0.9664	0.9290
enron1	Logistic Regression	Bernoulli	0.9539	0.9156	0.9463	0.9307
enron2	Multinomial NB	BoW	0.9414	0.8446	0.9615	0.8993
enron2	Bernoulli NB	Bernoulli	0.8536	0.8333	0.5769	0.6818
enron2	Logistic Regression	BoW	0.9540	0.9091	0.9231	0.9160
enron2	Logistic Regression	Bernoulli	0.9582	0.9231	0.9231	0.9231
enron4	Multinomial NB	BoW	0.9742	0.9748	0.9898	0.9822
enron4	Bernoulli NB	Bernoulli	0.9650	0.9537	1.0000	0.9763
enron4	Logistic Regression	BoW	0.9632	0.9513	1.0000	0.9751
enron4	Logistic Regression	Bernoulli	0.9687	0.9583	1.0000	0.9787

Table 2: Average Performance by Algorithm

Algorithm	Average Accuracy	Average Precision	Average Recall	Average F1-Score
Multinomial NB (BoW)	0.9514	0.9032	0.9592	0.9298
Bernoulli NB (Bernoulli)	0.8862	0.8841	0.7270	0.7899
Logistic Regression (BoW)	0.9563	0.9183	0.9632	0.9400
Logistic Regression (Bernoulli)	0.9603	0.9323	0.9565	0.9442

Best Overall Performance: Logistic Regression (Bernoulli) - 0.9603

Table 3: Best Algorithm per Dataset

Dataset	Best Algorithm	Accuracy
enron1	Logistic Regression (Bernoulli)	0.9539
enron2	Logistic Regression (Bernoulli)	0.9582
enron4	Multinomial NB (BoW)	0.9742

Table 4: Hyperparameters - Logistic Regression

Dataset	Representation	Learning Rate (η)	Regularization (λ)
enron1	BoW	0.1000	0.0000
enron1	Bernoulli	0.1000	0.0000
enron2	BoW	0.1000	0.0000
enron2	Bernoulli	0.1000	0.0000
enron4	BoW	0.1000	0.0010
enron4	Bernoulli	0.1000	0.0000

Additional Parameters:

- Max iterations (final training): 2000
- Train/validation split: 70/30 (stratified)
- Prediction threshold: 0.5

Table 5: Hyperparameters - Naive Bayes

Algorithm	Parameter	Value
Multinomial NB	Laplace smoothing (α)	1.0
Bernoulli NB	Laplace smoothing (α)	1.0

Naive Bayes Hyperparameters

Multinomial Naive Bayes

- **Laplace smoothing (α):** 1.0
  - Used in probability calculation: P(word|class) = (count + α) / (total + α × vocab_size)

Bernoulli Naive Bayes

- **Laplace smoothing (α):** 1.0
  - Used in probability calculation: P(word present|class) = (count + α) / (total + 2α)

Logistic Regression Hyperparameters

Tuned Hyperparameters (via grid search):

Learning Rate (η):

- Search space: [0.001, 0.01, 0.1]
- Selected: **0.1** for all datasets and representations

Regularization Strength (λ):

- Search space: [0.0, 0.001, 0.01, 0.1]
- Selected:
    - enron1 BoW: **0.0**
    - enron1 Bernoulli: **0.0**
    - enron2 BoW: **0.0**
    - enron2 Bernoulli: **0.0**
    - enron4 BoW: **0.001**
    - enron4 Bernoulli: **0.0**

Fixed Hyperparameters (not tuned):

Max Iterations:

- Hyperparameter tuning: 1000 iterations
- Final training: 2000 iterations

Train/Validation Split:

- 70% training, 30% validation (for hyperparameter selection)
- Stratified split to maintain class proportions

Prediction Threshold:

- 0.5 (classify as spam if P(spam|email) ≥ 0.5)

Weight Initialization:

- Random normal distribution: mean=0, std=0.01

Bias Initialization:

- 0.0

Optimization Algorithm:

- Gradient ascent (not a hyperparameter but worth noting)

Regularization Type:

- L2 (Ridge) regularization
- Applied to weights only, NOT bias

Data Preprocessing Hyperparameters

Vocabulary Building:

- **Minimum word frequency:** 2 (words appearing < 2 times are excluded)

Text Cleaning:

- Stopwords: NLTK English stopwords
- Minimum word length: 2 characters
- Character set: lowercase a-z only

Summary for Report

Table: Hyperparameters Used

Component	Hyperparameter	Value
Multinomial NB	Laplace smoothing α	1.0
Bernoulli NB	Laplace smoothing α	1.0
Logistic Regression	Learning rate η	0.1
Logistic Regression	Regularization λ	0.0 (mostly), 0.001 (enron4 BoW)
Logistic Regression	Max iterations	2000 (final training)
Logistic Regression	Train/val split	70/30 stratified
Logistic Regression	Prediction threshold	0.5
Preprocessing	Min word frequency	2

Analysis Questions

Question 1: Did Naive Bayes or Logistic Regression perform better? Why?
Answer: Logistic Regression performed better overall, achieving an average accuracy of 0.9583 (across both BoW and Bernoulli representations) compared to Naive Bayes's 0.9188 average.

Detailed Comparison:

- Multinomial NB: 0.9514 average accuracy
- Bernoulli NB: 0.8862 average accuracy
- Logistic Regression (BoW): 0.9563 average accuracy
- Logistic Regression (Bernoulli): 0.9603 average accuracy
-

Reasons for Superior Performance:

1. **Discriminative vs. Generative Modeling:** Logistic Regression is a discriminative classifier that directly models the conditional probability P(Y|X), optimizing the decision boundary between spam and ham emails. In contrast, Naive Bayes is a generative classifier that models P(X|Y) and uses Bayes' theorem to derive P(Y|X). For classification tasks, discriminative models typically achieve better performance because they focus directly on separating classes rather than modeling the entire data distribution.
2. **Feature Independence Assumption:** Naive Bayes makes the strong assumption that all features (words) are conditionally independent given the class label. This assumption is violated in text data where words frequently co-occur in patterns. For example, in spam emails, words like "click," "here," and "now" often appear together. Logistic Regression does not make this independence assumption and can learn correlations between features through its weight parameters.
3. **Gradient-Based Optimization:** Logistic Regression uses gradient ascent to iteratively optimize weights that maximize the log-likelihood function. With sufficient iterations (2000 in our implementation), this optimization process converges to decision boundaries that better separate the classes. Naive Bayes uses closed-form maximum likelihood estimates, which, while computationally efficient, cannot adapt as flexibly to complex decision boundaries.
4. **Consistent Performance:** Logistic Regression maintained robust performance across all datasets (range: 0.9518-0.9687), while Bernoulli Naive Bayes showed significant variability (range: 0.8399-0.9650), demonstrating LR's reliability across different data characteristics.

Question 2: Which combination of algorithm and data representation yielded the best performance? Why?
Answer: Logistic Regression with Bernoulli (binary) features achieved the best overall performance with an average accuracy of 0.9603.

Performance Breakdown by Dataset:

- enron1: LR + Bernoulli achieved 0.9539 (best for this dataset)
- enron2: LR + Bernoulli achieved 0.9582 (best for this dataset)
- enron4: Multinomial NB + BoW achieved 0.9742 (best), but LR + Bernoulli was close at 0.9687

Reasons for Superior Performance:

1. **Noise Reduction:** Binary feature representation reduces noise from word frequency variations. In spam detection, the critical information is often whether specific keywords appear (e.g., "lottery,", "click here") rather than how many times they occur. A spammer repeating "FREE" 50 times should not carry more weight than writing it once.
2. **Robustness to Adversarial Manipulation:** Spammers often employ techniques like keyword stuffing to evade frequency-based filters. Binary features make classifiers immune to such manipulation—whether a word appears once or a hundred times, it registers the same way in the feature vector.
3. **Effective Weight Learning:** Logistic Regression's gradient-based optimization effectively learns which binary features are strong spam indicators. It assigns large positive weights to spam-indicative words and negative weights to legitimate email terms, creating an optimal linear separator in the binary feature space.
4. **Balanced Precision-Recall Trade-off:** This combination achieved excellent recall (average 0.9565) while maintaining high precision (average 0.9323). This balance is crucial for spam filters—catching most spam (high recall) without excessive false positives that would frustrate users (high precision).
5. **Computational Efficiency:** Binary features result in sparser data representations, making both training and prediction computationally efficient while maintaining superior classification performance.

Question 3: Did Multinomial Naive Bayes perform better than Logistic Regression on the Bag of Words representation? Explain.

Answer: No. Multinomial Naive Bayes averaged 0.9514 accuracy while Logistic Regression on BoW averaged 0.9563 accuracy—a difference of 0.0049 in favor of LR.

Dataset-Level Analysis:

- enron1: LR wins (0.9518 vs. 0.9386) - 1.32 percentage point advantage
- enron2: LR wins (0.9540 vs. 0.9414) - 1.26 percentage point advantage
- enron4: Multinomial NB wins (0.9742 vs. 0.9632) - 1.10 percentage point advantage

Explanation:

1. **Natural Fit for Count Data:** Both algorithms are well-suited to word count features. Multinomial NB is specifically designed for count data and models word frequencies probabilistically, while LR can learn optimal weights for count features through gradient optimization. This explains the relatively close performance.
2. **Dataset-Specific Performance:** On enron4, Multinomial NB achieved exceptional performance (0.9742) because this dataset exhibits very clear class separation. When the independence assumption is approximately satisfied and classes are well-separated, NB's generative approach works effectively. The high F1-score (0.9822) confirms the quality of separation in this dataset.
3. **Consistency vs. Peak Performance:** While Multinomial NB achieved the single highest accuracy on enron4, Logistic Regression never significantly underperformed. LR's lowest accuracy (0.9518 on enron1) was higher than Multinomial NB's lowest (0.9386 on enron1), demonstrating more consistent performance across varying data characteristics.
4. **Feature Interaction Modeling:** Logistic Regression's advantage stems from its ability to model feature interactions through learned weights. For example, the combination of certain words appearing together can be more indicative of spam than their individual frequencies. LR captures this through its linear combination of weighted features, while Multinomial NB treats each word's frequency independently.
5. **Practical Significance:** The 0.49 percentage point difference, while small, is consistent across two of three datasets, suggesting a genuine advantage for the discriminative approach rather than random variation.

Question 4: Did Bernoulli Naive Bayes perform better than Logistic Regression on the Bernoulli representation? Explain.

Answer: No. Bernoulli Naive Bayes averaged 0.8862 accuracy while Logistic Regression on Bernoulli features averaged 0.9603 accuracy—a substantial 7.41 percentage point difference in favor of LR.

Dataset-Level Analysis:

- enron1: LR wins (0.9539 vs. 0.8399) - 11.40 percentage point gap
- enron2: LR wins (0.9582 vs. 0.8536) - 10.46 percentage point gap
- enron4: LR wins (0.9687 vs. 0.9650) - 0.37 percentage point gap

Explanation:

1. **Severe Independence Assumption Violations:** Bernoulli NB's assumption that word presence/absence is conditionally independent given the class is particularly limiting for spam detection. In reality, spam emails exhibit characteristic word co-occurrence patterns. For example, the presence of "lottery," "winner," and "claim" together is much more indicative of spam than each word independently. Bernoulli NB cannot model these patterns.
2. **Inability to Weight Feature Importance:** All features in Bernoulli NB are treated equally in terms of their structural contribution to the model. While probability estimates differ, the model cannot learn that some binary features (e.g., presence of "lottery") are far more important than others (e.g., presence of "the"). Logistic Regression learns feature importance through optimized weights.
3. **Critical Recall Deficiency:** Bernoulli NB exhibited severe recall problems on enron1 and enron2:
    - enron1: 0.6040 recall (missed 39.6% of spam emails)
    - enron2: 0.5769 recall (missed 42.3% of spam emails)
    - enron4: 1.0000 recall (caught all spam, but this dataset was easier)

In contrast, Logistic Regression maintained excellent recall across all datasets (0.9463-1.0000), demonstrating its ability to identify spam consistently.

4. **Feature Correlation Learning:** Logistic Regression's linear model w^T x + b allows it to learn both positive and negative correlations between features and the target class. Negative weights on legitimate email words (e.g., "meeting," "attached") help distinguish ham from spam. Bernoulli NB's probabilistic framework is less effective at capturing these nuanced relationships in binary feature space.
5. **Magnitude of Performance Gap:** The 7.41 percentage point average difference is substantial in classification tasks and indicates a fundamental limitation of Bernoulli NB for this problem. In practical deployment, missing over 40% of spam (as happened on two datasets) would be completely unacceptable for a spam filter.
6. **Statistical Significance:** The consistency of LR's advantage across all three datasets, with particularly large gaps on two of them, provides strong evidence that this is a systematic difference rather than random variation.

Implementation Notes:

Computational Complexity -

Achieved O(nd) complexity per iteration as specified:

- n = number of features (vocabulary size: approximately 3,000-5,000 words per dataset)
- d = number of training examples (approximately 1,000-3,000 emails per dataset)
- Total iterations: 2000 for final training

Implementation Strategy:

- Used vectorized NumPy operations throughout to avoid nested Python loops
- Gradient computation: gradient = (1/d) × X^T × (y - predictions) executes in O(nd) time
- Avoided $O(n^2d)$ complexity that would result from computing gradients feature-by-feature with nested loops

Key Technical Details -

Logistic Regression Implementation

1. Gradient Ascent for Log-Likelihood Maximization:

Update rule: $w = w + \eta \times [(1/d) \times X^T \times (y - \text{predictions}) - \lambda \times w]$

$b = b + \eta \times [(1/d) \times \text{sum}(y - \text{predictions})]$

- Used gradient ascent (addition) rather than descent (subtraction)
- Maximizes log-likelihood: log P(Y|X; w)
- Regularization term subtracts λw to penalize large weights

2. L2 Regularization:

- Applied only to weight vector w, not bias term b
- Prevents overfitting by penalizing large weight magnitudes
- Objective: maximize log-likelihood - $(\lambda/2) \times ||w||^2$

3. Numerical Stability:

- Sigmoid input clipping: $z \in [-250, 250]$ to prevent overflow in exp(-z)
- Probability clipping: $p \in [1e\text{-}15, 1\text{-}1e\text{-}15]$ to prevent log(0) errors
- These bounds prevent numerical instability without affecting model behavior

4. Weight Initialization:

- Weights initialized from normal distribution: $N(0, 0.01^2)$
- Small random values break symmetry while starting near zero
- Bias initialized to 0.0

Naive Bayes Implementation

1. Log-Space Calculations:

- All probability calculations performed in log-space
- Prevents underflow when multiplying many small probabilities
- Prediction: argmax_c [log P(Y=c) + Σ log P(X_j | Y=c)]

2. Laplace Smoothing (α = 1.0):

- **Multinomial NB:** P(word_j | class) = (count_j + 1) / (total_words + vocab_size)
- **Bernoulli NB:** P(word_j = 1 | class) = (doc_count_j + 1) / (total_docs + 2)
- Prevents zero probabilities for unseen words in training data
- Denominator adjusts for number of possible values (vocab_size for counts, 2 for binary)

3. **Feature Representation:**
   - **Multinomial:** Counts how many times each word appears in document
   - **Bernoulli:** Binary indicator (1 if word appears at least once, 0 otherwise)
   - Same vocabulary used for both representations

Data Preprocessing

1. **Vocabulary Construction:**
   - Built exclusively from training set to prevent data leakage
   - Minimum word frequency threshold: 2 occurrences
   - Removed stopwords using NLTK English stopwords list
   - Minimum word length: 2 characters
   - Vocabulary size: ~3,000-5,000 words per dataset
2. **Text Cleaning Pipeline:**
   - Convert to lowercase
   - Remove email headers (Subject, From, To, Date)
   - Remove URLs and email addresses
   - Strip HTML tags
   - Remove all non-alphabetic characters
   - Normalize whitespace
3. **Feature Matrix Generation:**
   - Training vocabulary applied to test set
   - Out-of-vocabulary words in test set ignored (treated as zero)
   - Ensures same feature dimensionality for train and test

Hyperparameter Tuning Process

1. **Grid Search Strategy:**
   - 70/30 stratified train-validation split
   - Grid: learning_rate $\in$ {0.001, 0.01, 0.1}, $\lambda \in$ {0.0, 0.001, 0.01, 0.1}
   - Total: 12 configurations tested per dataset/representation pair
   - Selected configuration with highest validation accuracy
2. **Final Training:**
   - Retrained on full training set using selected hyperparameters
   - 2000 iterations for convergence
   - Evaluated once on held-out test set

Software Engineering Practices

1. **Code Organization:**
   - Modular design with separate files for preprocessing, models, and evaluation
   - Reusable functions for common operations (metrics calculation, data loading)
   - Clear separation between training and evaluation logic
2. **Reproducibility:**
   - Random seeds set for train-validation splits
   - Deterministic vocabulary construction (sorted word lists)
   - Fixed hyperparameter search space
3. **Error Handling:**
   - Try-except blocks for file I/O operations
   - Validation of input dimensions before matrix operations
   - Graceful handling of edge cases (empty documents, zero counts)